Mảng
(Array)

Trịnh Tấn Đạt
Khoa CNTT - Đại Học Sài Gòn
Email: trinhtandat@sgu.edu.vn
Website: https://sites.google.com/site/ttdat88/

Nội dung

- Đặt vấn đề
- Định nghĩa mảng
- Mảng 1 chiều
- Các thao tác trên mảng 1 chiều (1D)
- Mảng 2 chiều (2D) và mảng nhiều chiều
- Các thao tác trên mảng 1 chiều (2D)
- Các ví dụ minh họa
- Bài Tập

Đặt vấn đề

❖ Ví dụ

- Chương trình cần lưu trữ 3 số nguyên?

  => Khai báo 3 biến int a1, a2, a3;

- Chương trình cần lưu trữ 100 số nguyên?

  => Khai báo 100 biến kiểu số nguyên!

- Người dùng muốn nhập n số nguyên?

  => Không thực hiện được!

❖ Giải pháp

- Kiểu dữ liệu mới cho phép lưu trữ một dãy các số nguyên và dễ dàng truy xuất.

Định nghĩa biến mảng

- Mảng (Array): là một kiểu dữ liệu có cấu trúc.
- Mảng là một nhóm các biến có cùng tên, cùng kiểu dữ liệu. Ví dụ: dãy các số nguyên, dãy các ký tự…
- Mảng có thể là một hoặc nhiều chiều.
- Kích thước được xác định ngay khi khai báo và không bao giờ thay đổi.
- Mỗi phần tử (mỗi biến) của mảng được truy xuất thông qua chỉ số.
- NNLT C luôn chỉ định một khối nhớ liên tục cho một biến kiểu mảng.

Mảng một chiều A

10	-2	5	0		

Mảng A có 6 ô nhớ (chứa tối đa 6 phần tử), có 4 phần tử gán giá trị, 2 ô nhớ trống

Định nghĩa biến mảng

- Ví dụ : mảng 2 chiều B kích thước 3x5 (có 3 dòng và 5 cột) chứa các số nguyên

1	6	-2	3	
0	37	43	51	
1	93	34	0	

- *Mảng B có 3x5=15 ô nhớ (chứa tối đa 15 phần tử), có 3x4=12 phần tử gán giá trị, 3 ô nhớ trống*

Khai báo biến mảng

- Cú pháp dùng cho mảng một chiều (**1D Array**):

  <Kiểu_Cơ_Sở> <Tên_Biến_Mảng>[<Số_Phần_Tử>];

- Lưu ý
  - **<Tên_Biến_Mảng>:** theo nguyên tắc đặt tên biến.
  - Phải xác định <số phần tử> cụ thể (hằng) khi khai báo. Ví dụ: 10 (mảng chứa 10 phần tử)
  - Bộ nhớ sử dụng phải ít hơn 64KB (65535 Bytes)
  - Một dãy liên tục có chỉ số từ 0 đến <tổng số phần tử>-1 (lưu ý: chỉ số bắt đầu từ 0)
- Ví dụ: int A [10]; // mảng số nguyên A chứa tối đa 10 phần tử

  float B[50]; //mảng kiểu float B chứa tối đa 50 phần tử

  double Array_C[1000] ; // mảng kiểu double Array_C chứa tối đa 1000 phần tử

Khai báo biến mảng

- Ví dụ: có thể dùng macro define xác định tổng số phần tử của mảng

```
#define  MAX 20   // MAX là hằng giá trị
int mangA[MAX]  // mảng số nguyên mangA chứa tối đa 20 phần tử
```

- Lưu ý khai báo dưới đây là SAI, không cho phép

```
int n = 100;
int M[n]; // SAI, chương trình sẽ báo lỗi
float N[k]; // SAI, chương trình sẽ báo lỗi
```

Khai báo biến mảng

- Xét ví dụ : int A[10];
- Ta đã khai báo (định nghĩa mảng) mảng số nguyên A có tổng số phần tử là 10.
- Các phần tử trong mảng chưa được gán giá trị.
- Tất cả các phần tử (nếu có) được gán bởi số nguyên.
- Mỗi phần tử của mảng có chỉ số từ 0 đến 9 (tổng cộng 10 phần tử)

Mảng A

Khai báo biến mảng

- Gọi N là số phần tử chứa trong mảng.
- Khi đó, chỉ số của mảng bắt đầu từ **0 đến N-1**
- Toán tử [] : toán tử lấy chỉ số trong mảng
- **Ví dụ:**

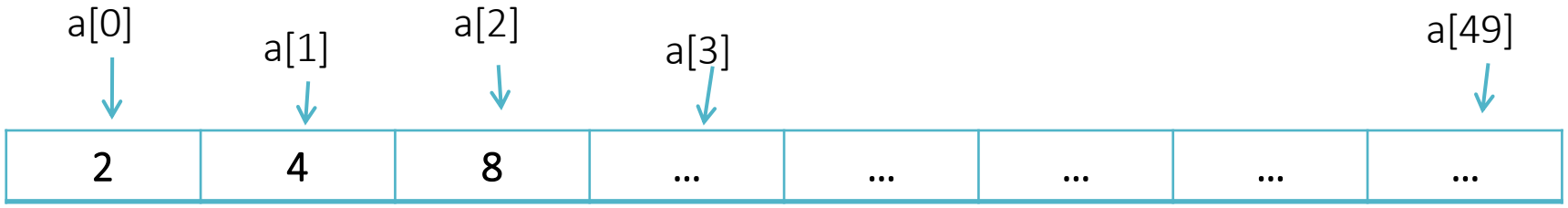
int a[50]; // khai báo mảng a chứa tối đa được 50 phần tử

Giả sử tạo mảng a chứa 3 phần tử là số nguyên (N = 3)

a[0] = 2;

a[1] = 4;

a[0] = 8;



Các chỉ số a[0] tại vị trí thứ nhất, a[1] tại vị trí thứ hai, a[2] tại vị trí thứ ba

Khai báo biến mảng

- Ví dụ: double B[10];
- Định nghĩa 10 biến kiểu double gồm: B[1], B[2], … , B[9]

Gán giá trị cho 4 phần tử đầu tiên từ 0 đến 3

B[0] = 10.5;

B[1] = 0.2;

B[2] = 9.5;

B[3] = 2.7;

Lưu trữ

- Khi mảng được định nghĩa,
  - Vùng nhớ cho các phần tử của mảng được cấp phát là các ô nhớ liền nhau.
  - Mỗi phần tử chiếm số bytes tùy thuộc kiểu dữ liệu
  - **sizeof(*<tên biến mảng>*);** → ***cho biết tổng*** số bytes của mảng.
    - Bộ nhớ sử dụng = <tổng số phần tử>*sizeof(<kiểu cơ sở>) Ví dụ: 10*sizeof(int) ~ 40 byte
- Ví dụ

```
int x[100];
sizeof(x) = ???
sizeof(x[0]) = sizeof(x[1]) = … = sizeof(x[99]) = ???
```

Lưu ý:

- Không thể dùng biến mảng như thông thường.

```
int a[6];
a = -2; // ERROR
cout<<a; // in ra gì???
```

- Mà phải dùng từng phần tử trong chúng. Mỗi phần tử được dùng như một biến đơn.

```
int a[6];
a[1] = -2;
cin>>a[0];  // nhập vào a[0]
cout<<a[0]*a[1];
```

Lưu ý

- Ta thường dùng phát biểu lặp để nhập hoặc truy xuất các phần tử của mảng.
- Đồng thời dùng thêm một biến kiểu nguyên, cho biết **số phần tử của mảng thực sự** đang được dùng.
- Ví dụ: Nhập vào mảng có 5 phần tử

```
int a[100];
int n;
cin>>n; // nhập vào n = 5
for (int i = 0; i<n; i++)
      cin>>a[i];
// in kết quả
for (int i = 0; i<n; i++)
      cout>>a[i];
```

Khởi tạo giá trị cho mảng

❖ Một số cách mà ngôn ngữ C cho phép khởi tạo giá trị của mảng

- Khởi tạo giá trị cho mọi phần tử của mảng (số phần tử = số giá trị)

```
int a[4] = {2912, 1706, 1506, 1904};
```

a[0] = 2912; a[1] = 1706; a[2] = 1506; a[3] = 1904

- Cấp phát 4 ô nhớ kiểu int, gán 4 giá trị cho các phần tử

Khởi tạo giá trị cho mảng

- Khởi tạo giá trị cho một số phần tử đầu mảng (số phần tử > số giá trị)

```
int a[4] = {2912, 1706};
```

  - Cấp phát 4 ô nhớ kiểu int, lần lượt gán giá trị cho 2 ô nhớ đầu tiên.

Khởi tạo giá trị cho mảng

- Tự động xác định số lượng phần tử (***Không định số phần tử***)

```
int a[] = {2912, 1706, 1506, 1904};
```

  - Cấp phát số phần tử = số giá trị được gán.
  - Cấp phát 4 ô nhớ kiểu int, lần lượt gán 4 giá trị cho các ô nhớ.

Khởi tạo giá trị cho mảng

- Khởi tạo giá trị 0 cho mọi phần tử của mảng

```
int a[4] = {0};
```

Lưu ý:

- Truy xuất đến một phần tử.

  - Cho mảng như sau: int a[4];

    Các truy xuất
    Hợp lệ: a[0], a[1], a[2], a[3]
    Không hợp lệ: a[-1], a[4], a[5], …
    => Cho kết thường không như mong muốn!

- **Không** được sử dụng phép gán thông thường mà phải gán trực tiếp giữa các phần tử tương ứng

```
int a[4] = {1, 4, 3, 5};
int b[4];
b = a;   // SAI
for (int i = 0; i < 4; i++)
        b[i] = a[i];
```

Một số lỗi thường gặp

- Khai báo không chỉ rõ số lượng phần tử

  int a[]; => int a[100];

- Số lượng phần tử liên quan đến biến hoặc hằng

  int n1 = 10; int a[n1]; => int a[10];

  const int n2 = 10; int a[n2]; => int a[10];

- Khởi tạo cách biệt với khai báo

  int a[4]; a = {2912, 1706, 1506, 1904};

  => int a[4] = {2912, 1706, 1506, 1904};

- Chỉ số mảng không hợp lệ

  int a[4];

  a[-1] = 1; a[10] = 0;

Biến mảng là tham số của hàm

- Tham số được khai báo như khai báo biến mảng.

Ví dụ: **void SapXepTang(int a[100]);**

- Tham số kiểu mảng truyền cho hàm chính là địa chỉ của phần tử đầu tiên của mảng
  - Có thể không cần xác định số phần tử của mảng
  - Mảng được chuyển tham chiếu (Mảng có thể thay đổi nội dung sau khi thực hiện hàm)

Ví dụ: Hàm xuất nội dung một mảng các số nguyên:

void xuatMang(int a[], int N);

Ví dụ: Hàm nhập nội dung một mảng các số nguyên:

void nhapMang(int a[], int &N); *// lưu ý khi nhập N trong hàm*

Biến mảng là tham số của hàm

- Ví dụ: Hàm tính tổng tất cả các phần tử trong mảng số nguyên A

[3, 1, 2] -> tổng các phần tử là 6

```
int tinhtongMang(int A[], int N);
```

- Ví dụ: Hàm sắp xếp mảng A theo thứ tự tăng dần (từ trái sang phải)

[3, 1, 2] -> [1, 2, 3]

```
void sapxepTang(int A[], int N);
```

- Ví dụ: Hàm xóa phần tử đầu tiên ra khỏi mảng A

[3, 1, 2] -> [1, 2]

```
void xoaDau(int A[], int &N);  // tại sao thêm tham chiếu trước N???
```

Biến mảng là tham số của hàm

- Lưu ý: Số lượng phần tử thực sự truyền qua biến khác

```
void SapXepTang(int a[100], int N);
void SapXepTang(int a[], int N);
```

- Lời gọi hàm

```
int  tinhtongMang(int A[], int N);
void sapxepTang(int A[], int N);
void xoaDau(int A[], int &N);
int main()
{
    int a[100], n;
    tinhtongMang(a,n);
    sapxepTang(a,n);
    xoaDau(a,n);
}
```

Các thao tác cơ bản trên 1D-Array

❖ Viết hàm thực hiện từng yêu cầu sau

- Nhập mảng/Xuất mảng
- Tìm kiếm một phần tử trong mảng
- Kiểm tra tính chất của mảng
- Tách mảng / Gộp mảng
- Tìm giá trị nhỏ nhất/lớn nhất của mảng
- Tính tổng giá trị trong mảng
- Sắp xếp mảng giảm dần/tăng dần
- Thêm/Xóa/Sửa một phần tử vào mảng

Nhập mảng

- Yêu cầu
- Cho phép nhập mảng a, số lượng phần tử n

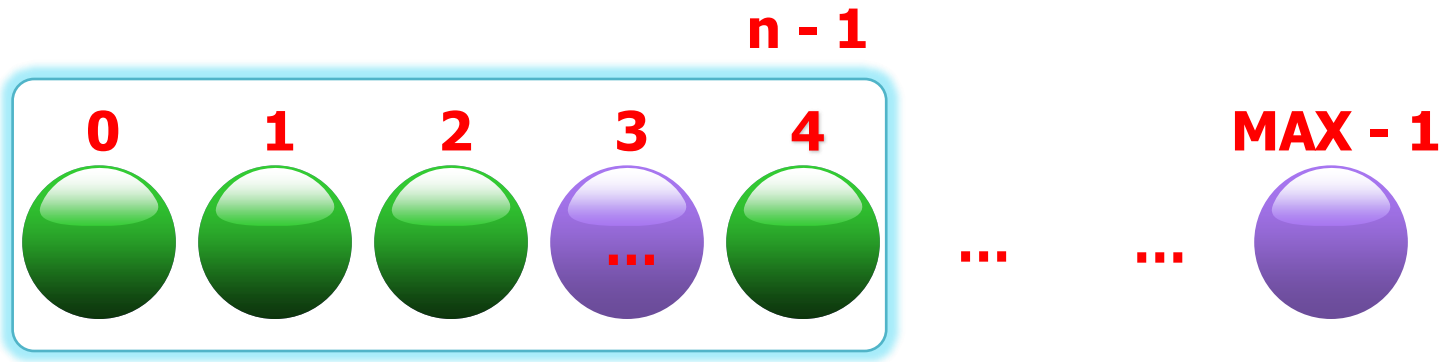
❖ Ý tưởng

- Khai báo một mảng 1 chiều có kích thước tùy ý (số lượng ô nhớ tối đa để lưu trữ các giá trị).

Ví dụ: int a[1000];

- Nhập số lượng phần tử thực sự n của mảng. (yêu cầu 0 < n < =1000)
- Nhập từng phần tử cho mảng từ chỉ số 0 đến n – 1.

Ví dụ:
#define MAX 1000



Hàm nhập mảng

```
void NhapMang(int a[], int &n)
{
   cout<<"Nhap so luong phan tu n:";
   cin>>n;

   for (int i = 0; i < n; i++)
   {
      cout<<"Nhap phan tu thu "<< i <<" :";
      cin>>a[i];
      // có thể dùng scanf để nhập
         // scanf("%d", &a[i]);
   }
}
```

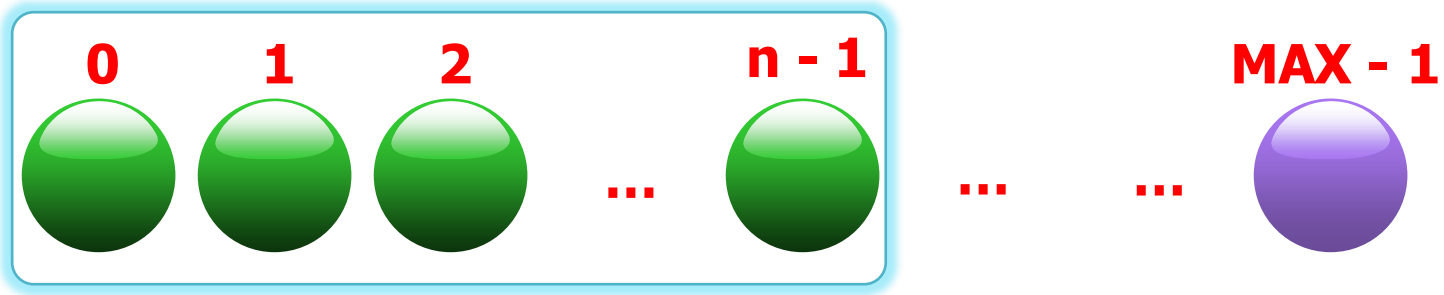
Xuất mảng

❖ Yêu cầu

- Cho trước mảng a, số lượng phần tử n. Hãy xuất nội dung mảng a ra màn hình.

❖ Ý tưởng

- Xuất giá trị từng phần tử của mảng từ chỉ số 0 đến n-1.



Hàm xuất mảng

```
void XuatMang(int a[], int n)
{
    cout<<"Cac phan tu cua mang:";

    for (int i = 0; i < n; i++)
        cout<<a[i]<<"\t";

    cout<<endl;
}
```

Ví dụ

- Chương trình hoàn chỉnh nhập và xuất mảng số nguyên

```
#include <iostream>
using namespace std;
void NhapMang(int a[], int &n); // prototype
void XuatMang(int a[], int n); // prototype
int main()
{
    int A[100];  // khai báo biến mảng
    int N; // khai báo số lượng phần tử trong mảng
    NhapMang(A,N);
    XuatMang(A,N);
    return 0;
}
```

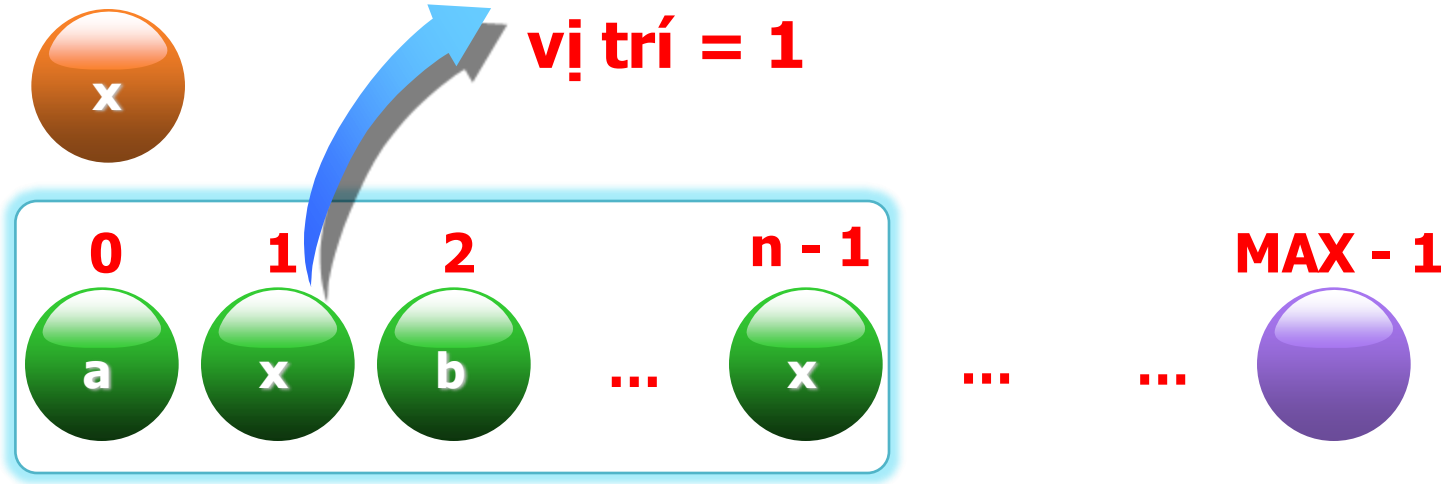
Tìm kiếm phần tử trong mảng

❖ Yêu cầu

- Tìm xem phần tử x có nằm trong mảng a kích thước n hay không? Nếu có thì nó nằm ở **vị trí đầu tiên** nào.

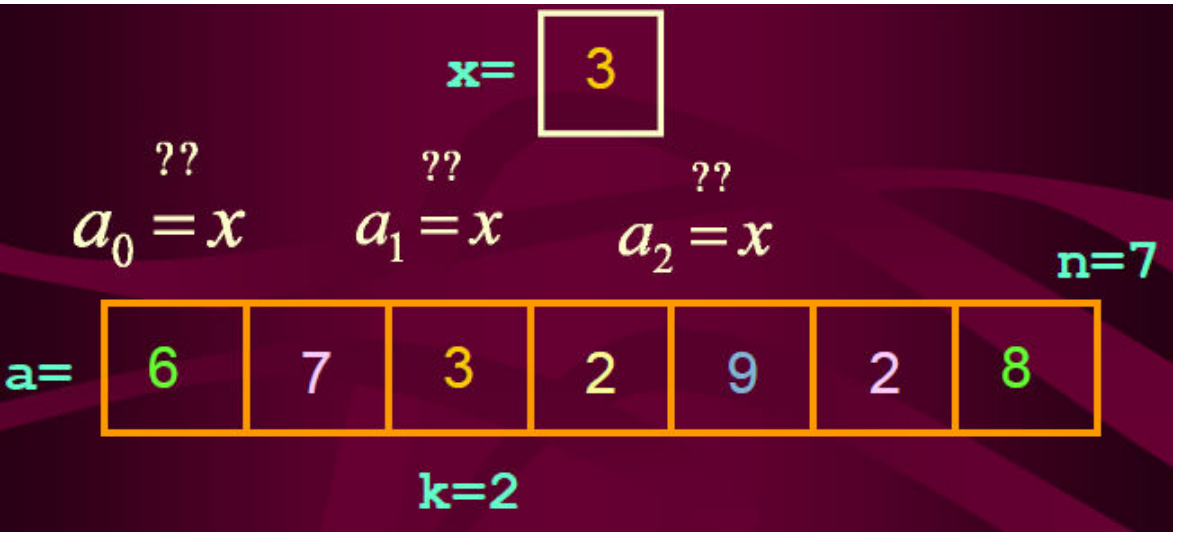
❖ Ý tưởng

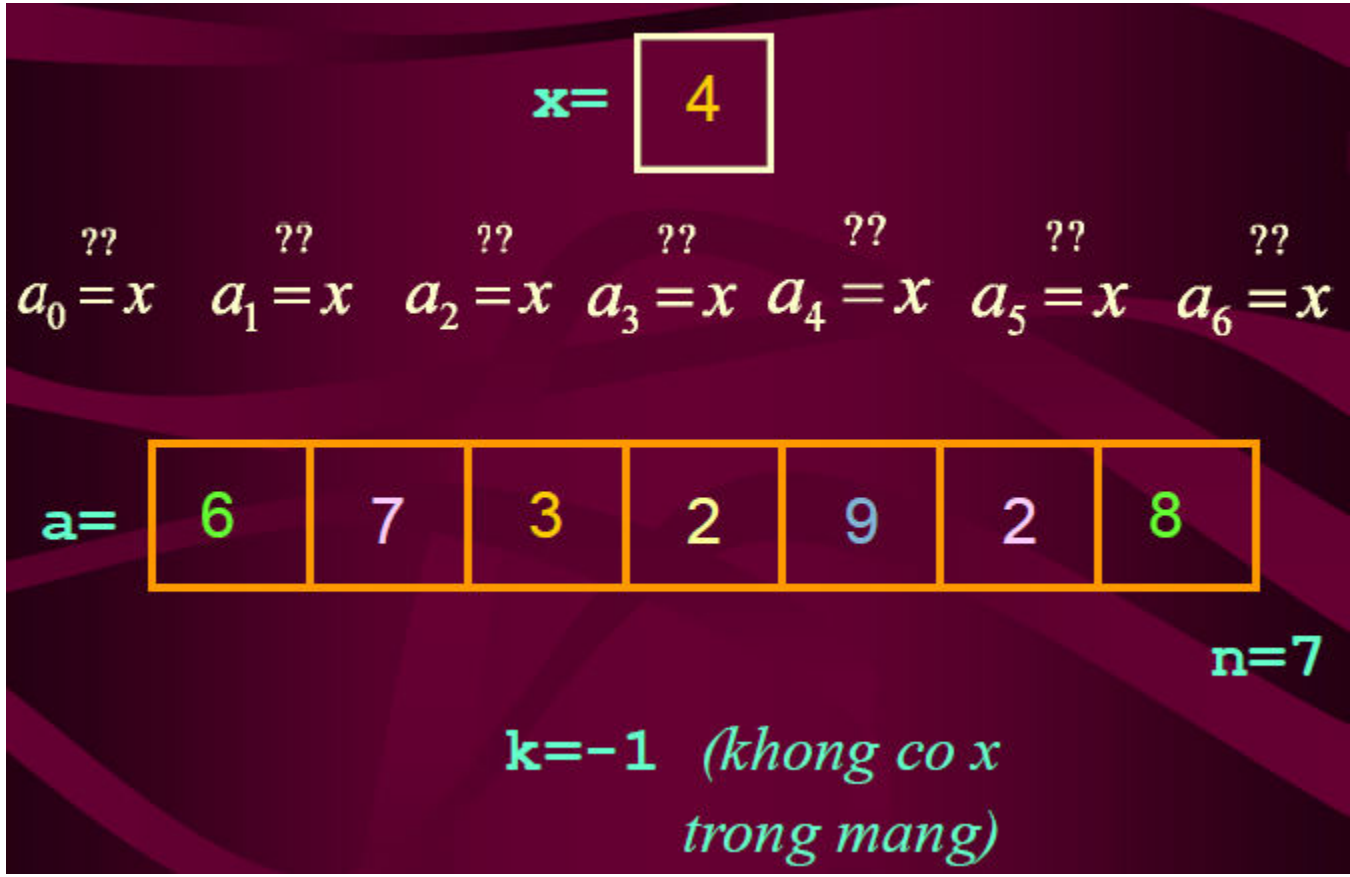
- Xét từng phần của mảng a. Nếu phần tử đang xét bằng x thì trả về vị trí đó. Nếu không tìm được thì trả về -1.



Tìm kiếm phần tử trong mảng

- Ví dụ:





Hàm Tìm Kiếm (dùng while)

```
int TimKiem(int a[], int n, int x)
{
    int pos = 0;

    while (pos < n && a[pos] != x)
        pos++;

    if (pos < n)
        return pos;

    return -1;
}
```

Hàm Tìm Kiếm (dùng for)

```
int TimKiem(int a[], int n, int x)
{
   for (int i= 0; i < n; i++)
      if (a[i] == x)
         return i;

   return -1;
}
```

```
#include <iostream>
using namespace std;
void NhapMang(int a[], int &n);
void XuatMang(int a[], int n);
int TimKiem(int a[], int n, int x);
int main()
{
    int A[100], N, x, k;
    NhapMang(A,N);
    XuatMang(A,N);
    cout<<"tim phan tu x:";
    cin>>x;
    k = TimKiem(A,N,x);
    if (k!=-1)
        cout<<"\n x tai vi tri:"<<k;
     else
        cout<<"\n khong tim thay";
    return 0;
}
```

Kiểm tra tính chất của mảng

❖ Yêu cầu

- Cho trước mảng a, số lượng phần tử n. Mảng a có phải là mảng toàn các số nguyên tố hay không?

❖ Ý tưởng

- Cách 1: Đếm số lượng số nguyên tố của mảng. Nếu số lượng này bằng đúng n thì mảng toàn nguyên tố.
- Cách 2: Đếm số lượng số không phải nguyên tố của mảng. Nếu số lượng này bằng 0 thì mảng toàn nguyên tố.
- Cách 3: Tìm xem có phần tử nào không phải số nguyên tố không. Nếu có thì mảng không toàn số nguyên tố $\Rightarrow$ **kỹ thuật đặt cờ hiệu**

Hàm kiểm tra SNT

```
int LaSNT(int n)
{
    int i, dem = 0;
    for (i = 1; i <= n; i++)
        if (n%i == 0)
            dem++;

    if (dem == 2)
        return 1;
    return 0;
}
```

Hàm Kiểm Tra (Cách 1)

```
int KiemTra_C1(int a[], int n)
{
    int dem = 0;

    for (int i = 0; i < n; i++)
        if (LaSNT(a[i]) == 1)
            dem++;

    if (dem == n)
        return 1;
    return 0;
}
```

Hàm Kiểm Tra (Cách 2)

```
int KiemTra_C2(int a[], int n)
{
    int dem = 0;

    for (int i = 0; i < n; i++)
        if (LaSNT(a[i]) == 0)
            dem++;

    if (dem == 0)
        return 1;
    return 0;
}
```

Hàm Kiểm Tra (Cách 3)

```
int KiemTra_C3(int a[], int n)
{
   for (int i = 0; i < n ; i++)
      if (LaSNT(a[i]) == 0) //không là SNT
         return 0;

   return 1;
}
```

Kỹ thuật đặt cờ hiệu

```
int KiemTra(int a[], int n)
{
    int flag = 1; // thảo đk
    for (int i = 0; i < n ; i++)
     {
        if (LaSNT(a[i]) == 0) //không là SNT
        {
             flag = 0;
             break;
         }
     }
    if (flag == 1)
        return 1;
     else
        return 0;
}
```

Tách các phần tử thỏa điều kiện

❖ Yêu cầu

- Cho trước mảng a, số lượng phần tử na. Tách các số nguyên tố có trong mảng a vào mảng b.
- Mảng na không thay đổi.

❖ Ý tưởng

- Duyệt từ phần tử của mảng a, nếu đó là số nguyên tố thì đưa vào mảng b.

Tách các phần tử thỏa điều kiện

```
void TachSNT(int a[], int na, int b[], int &nb)
{
    nb = 0;

    for (int i = 0; i < na; i++)
        if (LaSNT(a[i]) == 1)
        {
            b[nb] = a[i];
            nb++;
        }
}
```

Tách mảng thành 2 mảng con

❖ Yêu cầu

- Cho trước mảng a, số lượng phần tử na. Tách mảng a thành 2 mảng b (chứa số nguyên tố) và mảng c (các số còn lại).
- Mảng na không thay đổi.

❖ Ý tưởng

- Cách 1: viết 1 hàm tách các số nguyên tố từ mảng a sang mảng b và 1 hàm tách các số không phải nguyên tố từ mảng a sang mảng c.
- Cách 2: Duyệt từ phần tử của mảng a, nếu đó là số nguyên tố thì đưa vào mảng b, ngược lại đưa vào mảng c.

Hàm Tách 2 Mảng

```
void TachSNT2(int a[], int na,int b[], int &nb, int c[], int &nc)
{
   nb = 0;
   nc = 0;

   for (int i = 0; i < na; i++)
      if (LaSNT(a[i]) == 1)
      {
         b[nb] = a[i]; nb++;
      }
      else
      {
         c[nc] = a[i]; nc++;
      }
}
```

Hàm Gộp Mảng

❖ Yêu cầu

- Cho trước mảng a, số lượng phần tử na và mảng b số lượng phần tử nb. Gộp 2 mảng trên theo tứ tự đó thành mảng c, số lượng phần tử nc.

  nc = na+nb

❖ Ý tưởng

- Chuyển các phần tử của mảng a sang mảng c

  => nc = na

- Tiếp tục đưa các phần tử của mảng b sang mảng c

  => nc = nc + nb

Hàm Gộp Mảng

```
void GopMang(int a[], int na, int b[], int nb, int c[], int &nc)
{
   nc = 0;

   for (int i = 0; i < na; i++)
   {
      c[nc] = a[i]; nc++; // c[nc++] = a[i];
   }

   for (int i = 0; i < nb; i++)
   {
      c[nc] = b[i]; nc++; // c[nc++] = b[i];
   }
}
```

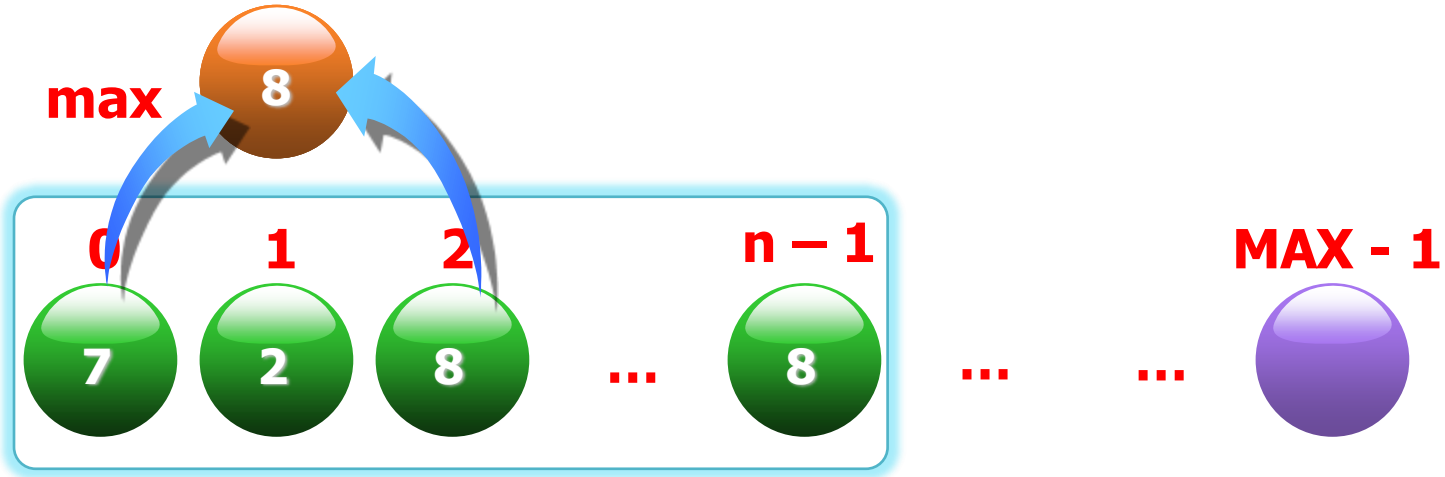
Tìm giá trị lớn nhất của mảng

❖ Yêu cầu

- Cho trước mảng a có n phần tử. Tìm giá trị lớn nhất trong a (gọi là max)

❖ Ý tưởng: Kỹ thuật đặt lính canh

- Giả sử giá trị max hiện tại là giá trị phần tử đầu tiên a[0]
- Lần lượt kiểm tra các phần tử còn lại để cập nhật max.



Hàm tìm Max

```
int TimMax(int a[], int n)
{
   int max = a[0];  // max là lính canh

   for (int i = 1; i < n; i++)
      if (a[i] > max)
         max = a[i];

   return max;
}
```

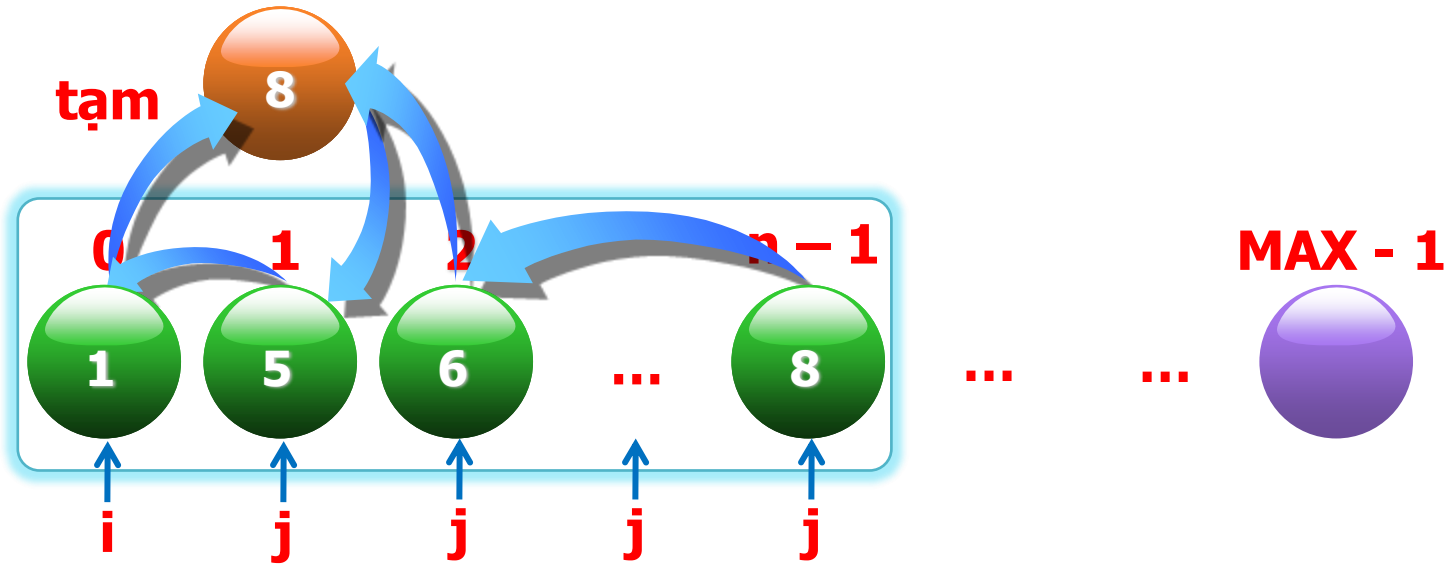
Hàm Sắp Xếp Tăng

❖ Yêu cầu

- Cho trước mảng a kích thước n. Hãy sắp xếp mảng a đó sao cho các phần tử có giá trị tăng dần.

❖ Ý tưởng : dựa vào thuật toán đơn giản **interchange sort**

- Sử dụng 2 biến i và j để so sánh tất cả cặp phần tử với nhau và hoán vị các cặp nghịch thế (sai thứ tự).



Hàm hoán vị 2 phần tử

```
void SapXepTang(int a[], int n)
{
    int i, j;

    for (i = 0; i < n - 1; i++)
    {
        for (j = i + 1; j < n; j++)
        {
            if (a[i] > a[j])
                HoanVi(a[i], a[j]);
        }
    }
}
```

```
void HoanVi(int &x, int &y)
{
    int tam = x; x = y; y = tam;
}
```

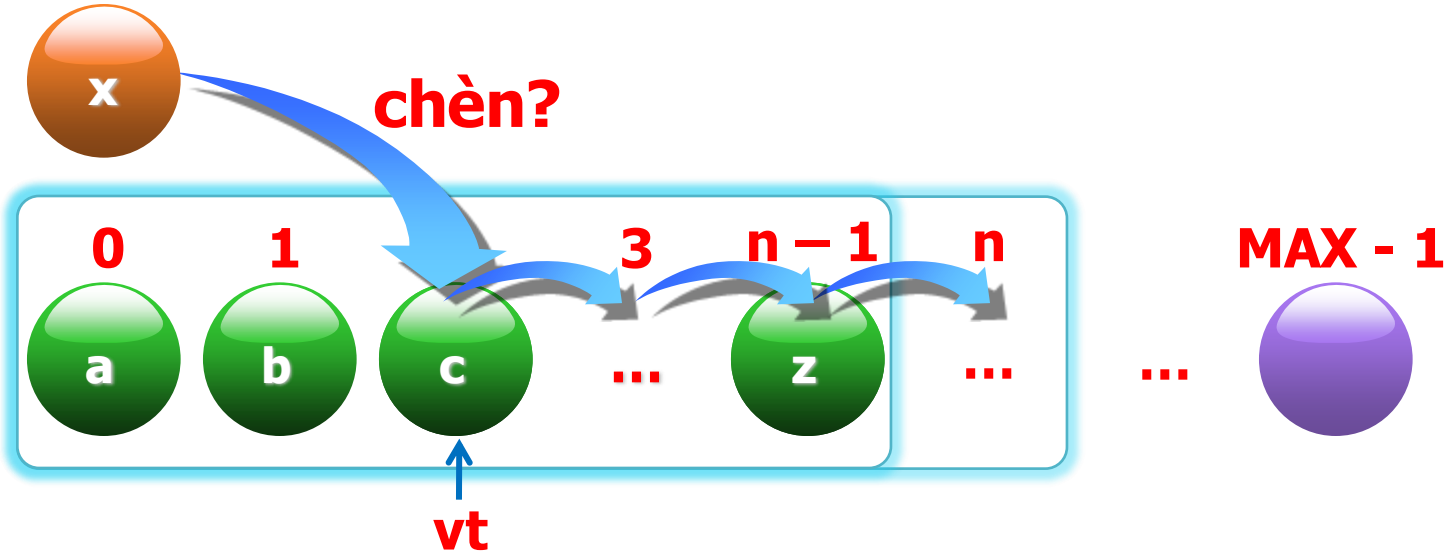
Thêm một phần tử vào mảng

❖ Yêu cầu

- Thêm phần tử x vào mảng a kích thước n tại vị trí vt.

❖ Ý tưởng

- "Đẩy" các phần tử bắt đầu tại vị trí vt sang phải 1 vị trí.
- Đưa x vào vị trí vt trong mảng.
- Tăng n lên 1 đơn vị.



Thêm một phần tử vào mảng

```
void Them(int a[], int &n, int vt, int x)
{
    if (vt >= 0 && vt <= n)
    {
        for (int i = n; i > vt; i--)
            a[i] = a[i - 1];

        a[vt] = x;
        n++;
    }
}
```

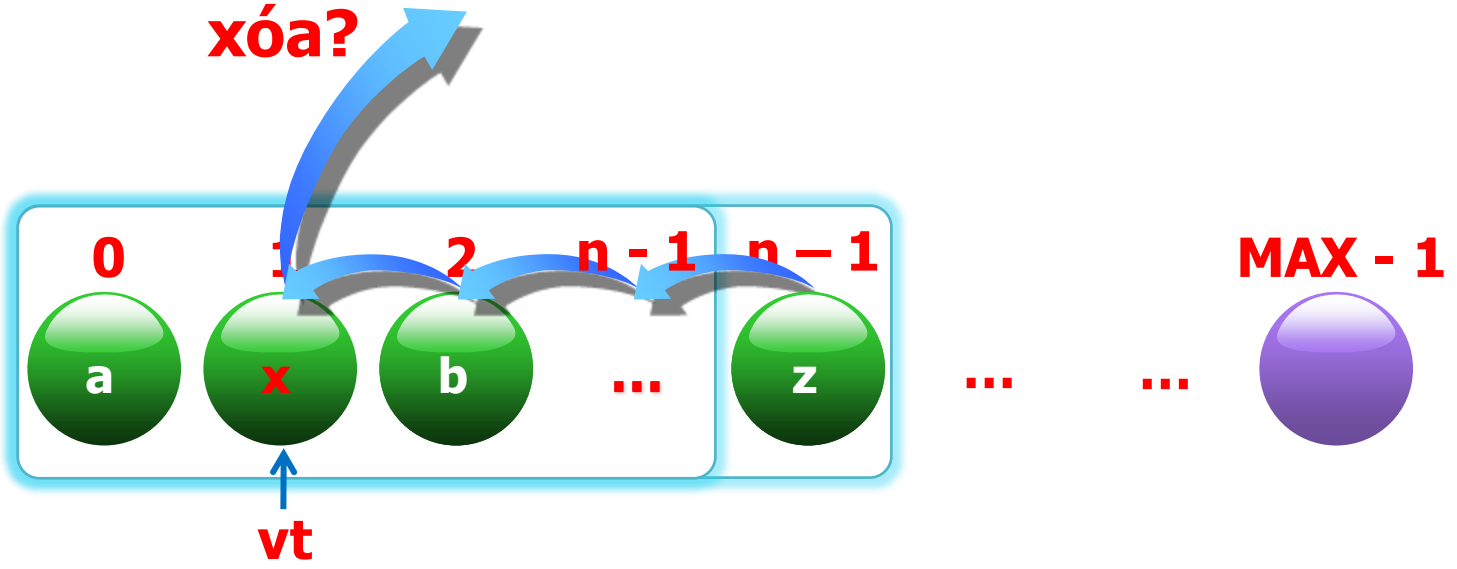
Xóa một phần tử trong mảng

❖ Yêu cầu

- Xóa một phần tử trong mảng a kích thước n tại vị trí vt

❖ Ý tưởng

- "Kéo" các phần tử bên phải vị trí vt sang trái 1 vị trí.
- Giảm n xuống 1 đơn vị.



Xóa một phần tử trong mảng

```
void XoaVT(int a[], int &n, int vt)
{
   if (vt >= 0 && vt < n)
   {
      for (int i = vt; i < n - 1; i++)
         a[i] = a[i + 1];

      n--;
   }
}
```

Xóa một phần tử x trong mảng

❖ Yêu cầu

- Xóa một phần tử có giá trị x ra khỏi mảng a có kích thước n.

❖ Ý tưởng

- Áp dụng hàm tìm kiếm phần tử trong mảng tìm vị trí của x là k

  int k = TimKiem(a,n,x);

- Nếu k khác = -1 (// tìm được x trong mảng a)

  Gọi hàm: **XoaVT(a,n,k);**

  Ngược lại (//không tìm thấy x trong mảng)

  Bỏ qua

Xóa một phần tử x trong mảng

```
void XoaX(int a[], int &n, int x)
{
   int k;
    k = TimKiem(a,n,x);
   if (k !=-1)
      XoaVT(a,n,k);
}
```

Xóa tất cả phần tử x trong mảng

```
void XoaX(int a[], int &n, int x)
{
   int k;
   do
   {
      k = TimKiem(a,n,x);
      if (k !=-1)
         XoaVT(a,n,k);
   }while(k!=-1);
}
```

Các ví dụ minh họa

1. Nhập vào một mảng 1 chiều A các số thực có kích thước N. Nhập vào giá trị x. Hãy tìm một vị trí trong mảng A mà giá trị tại vị trí đó là giá trị "gần giá trị x nhất".

```
#include <iostream>
#include <stdlib.h>
#include <math.h>   // fabs
#include <float.h>  // FLT_MAX
using namespace std;
int nhapmang(float a[], int &);
int xuatmang(float a[], int );
int vitrigannhat(float a[], int, float);
```

```
int nhapmang(float a[], int &N)
{
        cin >> N;
        for (int i=0;i<N;i++)
                cin>>a[i];
        return 0;
}
```

```
int xuatmang(float a[],int N)
{
        for (int k=0;k<N;k++)
                cout<< a[k] <<"  ";
        cout <<endl;
        return 0;
}
```

Các ví dụ minh họa

- Cách 1:

```
int vitrigannhat(float a[], int N, float x)
{
	float min_value = FLT_MAX; // so lon nhat kieu so FLOAT
	int min_index = -1;
	for (int i=0;i<N;i++)
	{
		if (fabs(a[i]-x)<min_value)
		{
			min_value = fabs(a[i]-x);
			min_index = i;
		}
	}
	cout <<"min distance:" << min_value <<endl;
	return min_index;
}
```

```
int main()
{
	float a[100], x;
	int N;
	nhapmang(a,N);
	xuatmang(a,N);
	cout<<"nhap x:";
	cin >>x;
	int k = vitrigannhat(a,N,x);
	cout <<"\nvi tri gan nhat:" << k ;
	system("pause");
	return 0;
}
```

Các ví dụ minh họa

- Cách 2:

Step 1: tính trị tuyệt đối khoảng cách giữa các phần tử a[i] với x : **fabs(a[i]-x)**

Step 2: tìm giá trị nhỏ nhất trong mảng sau khi thực hiện step 1 **fabs(a[i]-x).** Khi đó vị trí ứng với giá trị nhỏ nhất là vị trí cần tìm

Các ví dụ minh họa

2. Nhập vào mảng 1 chiều A các số nguyên và sắp xếp mảng A trên theo quy luật:

- Các số chẵn (nếu có) ở đầu mảng và có thứ tự giảm dần.
- Các số lẻ (nếu có) ở cuối mảng và có thứ tự tăng dần.

5	4	3	6	9	2

Nhân các số chẵn với -1

5	- 4	3	-6	9	-2

Sắp xếp tăng

-6	- 4	-2	3	5	9

Nhân các số chẵn với -1

6	4	2	3	5	9

Các ví dụ minh họa

```
void sapxepchangiamletang(int a[], int N)
{
	// nhan -1 cho cac so chan
	for (int i=0;i<N;i++)
	{
		if (a[i]%2==0)
			a[i]= a[i]*-1;
	}
	sapxeptang(a,N); // viet ham sap xep tang
	// nhan -1 cho cac so chan
	for (int i=0;i<N;i++)
	{
		if (a[i]%2==0)
			a[i]= a[i]*-1;
	}
}
```

Các ví dụ minh họa

3. Nhập mảng 1 chiều N số nguyên dương. Tìm số nguyên tố nhỏ nhất (ở ngoài mảng) thỏa điều kiện lớn hơn mọi giá trị có trong mảng.

```
int findmaxarray(int a[], int N)
{
        int maxvalue = INT_MIN;
        for (int i=0;i<N;i++)
        {
            if (a[i]>maxvalue)
                maxvalue = a[i];
        }
        return maxvalue;
}
```

```
int checkprime(int K)
{
    if (K <2) return 0;
     for (int i=2;i<=K/2;i++)
     {
          if (K%i==0)
          return 0;
      }
       return 1;
}
```

Các ví dụ minh họa

```
int findminprime(int a[],int N)
{
        int maxvalue = findmaxarray(a,N);
        int nextnum = maxvalue +1;
        while(true)
        {
                if(checkprime(nextnum)==1)
                        break;
                else
                        nextnum++;
        }
        return nextnum;
}
```

Ôn Tập

- Viết chương trình hoàn chỉnh (có hàm main) cho tất cả thao tác cơ bản trên mảng một chiều.
- Yêu cầu: Nhập mảng từ bàn phím. Xuất mảng và các kết quả theo yêu cầu ra màn hình.

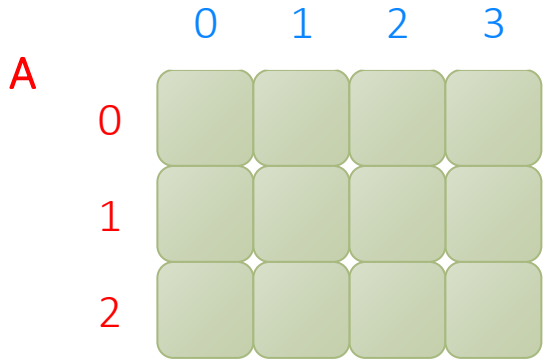
Mảng hai chiều

- Cú pháp dùng cho mảng hai chiều (**2D-Array**):

<Kiểu_Cơ_Sở> <Tên_Biến_Mảng>[<N1>] [<N2>];

  - N1, N2: số lượng phần tử mỗi chiều

- Ví dụ: int A[3][4]; // mảng 2 chiều A có 3 dòng và 4 cột



Ví dụ:
int a[10][20], b[10][20];
float c[5][10];
char d[10][20];

Ví dụ

- Ví dụ : mảng 2 chiều B kích thước 3x5 (có 3 dòng và 5 cột) chứa các số nguyên

  int B[3][5];

	Cột 0	Cột 1	Cột 2	Cột 3	
Dòng 0	1	6	-2	3	
Dòng 1	0	37	43	51	
Dòng2	1	93	34	0	

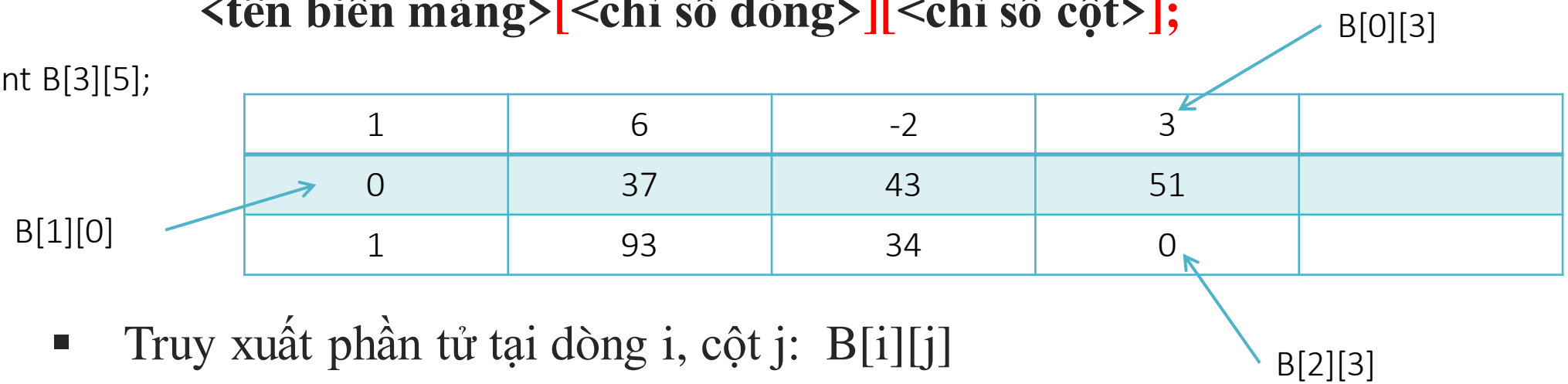
- *Mảng B có 3x5=15 ô nhớ (chứa tối đa 15 phần tử), có 3x4=12 phần tử gán giá trị, 3 ô nhớ trống*

Truy xuất đến một phần tử

- Thông qua chỉ số: truy xuất thông tên mảng và chỉ số tại dòng và cột tương ứng

<tên biến mảng>[<chỉ số dòng>][<chỉ số cột>];

int B[3][5];



1	6	-2	3	
0	37	43	51	
1	93	34	0	

- Truy xuất phần tử tại dòng i, cột j: B[i][j]
- Các truy xuất
  - Hợp lệ: B[0][0], B[0][1], …, B[2][2], B[2][3]
  - Không hợp lệ: B[-1][0], B[4][4], B[3][7]

Gán dữ liệu kiểu mảng

- KHÔNG được sử dụng phép gán thông thường mà phải gán trực tiếp giữa các phần tử.

```
<biến mảng đích> = <biến mảng nguồn>; //sai
<biến mảng đích>[<giá trị cs1>][giá trị cs2] = <giá trị>;
```

- Ví dụ:

```
int a[5][10], b[5][10];

b = a;   // SAI

int i, j;
for (i = 0; i < 5; i++)
   for (j = 0; j < 10; j++)
      b[i][j] = a[i][j];
```

Khởi tạo giá trị cho mảng 2 chiều

- Khởi tạo giá trị cho một các phần tử trong mảng 2 chiều (có 5 dòng , 5 cột)

```
int A[5][5] = {{1, 0, 0, 0, 1},
               {1, 0, 0, 0, 1},
               {1, 1, 1, 1, 1},
               {1, 0, 0, 0, 1},
               {1, 0, 0, 0, 1}};
```

- int [3][4] = {{1},{4,5}};
  - Ý nghĩa là a[0][0] = 1; a[1][0]=4; a[1][1] = 5; tất cả các vị trí khác bằng 0
- int [3][4] = {};
  - Ý nghĩa là tất cả các phần tử đều bằng 0

Truyền mảng cho hàm

❖ Truyền mảng cho hàm

- Tham số kiểu mảng trong khai báo hàm giống như khai báo biến mảng

```
void NhapMangHaiChieu(int a[50][100]);
```

- Tham số kiểu mảng truyền cho hàm chính là địa chỉ của phần tử đầu tiên của mảng.
  - Có thể bỏ số lượng phần tử chiều thứ 1.
  - Mảng có thể thay đổi nội dung sau khi thực hiện hàm.

```
void NhapMangHaiChieu(int b[][100]);
```

Truyền mảng cho hàm

❖ Truyền mảng cho hàm

- Số lượng phần tử thực sự truyền qua biến khác

```
void XuatMangHaiChieu(int a[50][100], int m, int n);
void XuatMangHaiChieu (int a[][100], int m, int n);
```

m là số dòng, n là số cột

- Lời gọi hàm

```
void NhapMangHaiChieu(int a[][100], int &m, int &n);
void XuatMangHaiChieu(int a[][100], int m, int n);
void main()
{
   int a[50][100], m, n;
   NhapMangHaiChieu (a, m, n);
   XuatMangHaiChieu (a, m, n);
}
```

Các thao tác cơ bản trên 2D-Array

❖ Viết chương trình con/hàm thực hiện các yêu cầu sau

- Nhập mảng
- Xuất mảng
- Tìm kiếm một phần tử trong mảng
- Kiểm tra tính chất của mảng
- Tính tổng các phần tử trên dòng/cột/toàn mảng
- Tìm giá trị nhỏ nhất/lớn nhất của mảng
- …

Một số quy ước

- Kiểu dữ liệu

```
#define MAXD 50
#define MAXC 100
```

- Các chương trình con
  - Hàm void HoanVi(int x, int y): hoán vị giá trị của hai số nguyên.
  - Hàm int LaSNT(int n): kiểm tra một số có phải là số nguyên tố. Trả về 1 nếu n là số nguyên tố, ngược lại trả về 0.

Thủ tục HoanVi & Hàm LaSNT

```
void HoanVi(int &x, int &y)
{
   int tam = x; x = y; y = tam;
}

int LaSNT(int n)
{
   int i, dem = 0;
   for (i = 1; i <= n; i++)
      if (n%i == 0)
         dem++;

   if (dem == 2)
      return 1;
   return 0;
}
```

Nhập Mảng 2 chiều

❖ Yêu cầu

- Cho phép nhập mảng a, m dòng, n cột

❖ Ý tưởng

- Cho trước một mảng 2 chiều có dòng tối đa là MAXD, số cột tối đa là MAXC.
- Nhập số lượng phần tử thực sự m, n của mỗi chiều.
- Nhập từng phần tử từ [0][0] đến [m-1][n-1].

Hàm Nhập Mảng 2 chiều

```
void NhapMangHaiChieu(int a[][MAXC], int &m, int &n)
{
   cout<<"Nhap so dong, so cot cua mang:";
   cin>>m>>n;

   int i, j;
   for (i=0; i<m; i++)
      for (j=0; j<n; j++)
      {
         cout<<"Nhap phan tu:";
         cin>>a[i][j];
      }
}
```

Hàm Nhập Mảng 2 chiều

```
void NhapMangHaiChieu(int a[][MAXC], int &m, int &n)
{
   printf("Nhap so dong, so cot cua ma tran: ");
   scanf("%d %d", &m, &n);

   int i, j;
   for (i=0; i<m; i++)
      for (j=0; j<n; j++)
      {
         printf("Nhap a[%d][%d]: ", i, j);
         scanf("%d", &a[i][j]);
      }
}
```

Xuất Mảng 2 chiều

❖ Yêu cầu

- Cho phép in ra màn hình mảng a, m dòng, n cột

❖ Ý tưởng

- Xuất giá trị từng phần tử của mảng 2 chiều từ dòng có 0 đến dòng m-1, mỗi dòng xuất giá giá trị của cột 0 đến cột n-1 trên dòng đó.

Hàm Xuất Mảng 2 chiều

```
void XuatMangHaiChieu(int a[][MAXC], int m, int n)
{
    int i, j;
    for (i=0; i<m; i++)
    {
        for (j=0; j<n; j++)
            cout<<a[i][j];

        cout<<endl;
    }
}
```

Ví dụ:

- Chương trình nhập và xuất mảng 2 chiều

```
#include <iostream>
#define MAXD 50
#define MAXC 100
void NhapMangHaiChieu(int a[][MAXC], int &m, int &n);
void XuatMangHaiChieu(int a[][MAXC], int m, int n);
int main()
{
    int  a[MAXD][MAXC];
    int  n,m;
    NhapMangHaiChieu(a,m,n);
    XuatMangHaiChieu(a,m,n);
     return 0;
}
// hàm nhập mảng
// hàm xuất mảng
```

Tìm kiếm một phần tử trong Mảng

❖ Yêu cầu

- Tìm xem phần tử x có nằm trong mảng a kích thước mxn hay không?

❖ Ý tưởng

- Duyệt từng phần của mảng a. Nếu phần tử đang xét bằng x thì trả về có (1), ngược lại trả về không có (0).

Hàm Tìm Kiếm

```
int TimKiem(int a[][MAXC], int m, int n, int x)
{
   int i, j;
   for (i=0; i<m; i++)
      for (j=0; j<n; j++)
         if (a[i][j] == x)
         {
            cout<<"tai dong:"<<i;
            cout<<"tai cot:"<<j;
            return 1;
         }
   return 0;
}
```

Kiểm tra tính chất của mảng

❖ Yêu cầu

- Cho trước mảng a kích thước mxn. Mảng a có phải là mảng toàn các số nguyên tố hay không?

❖ Ý tưởng

- Cách 1: Đếm số lượng số nguyên tố của mảng. Nếu số lượng này bằng đúng mxn thì mảng toàn nguyên tố.
- Cách 2: Đếm số lượng số không phải nguyên tố của mảng. Nếu số lượng này bằng 0 thì mảng toàn nguyên tố.
- Cách 3: Tìm xem có phần tử nào không phải số nguyên tố không. Nếu có thì mảng không toàn số nguyên tố.

Hàm Kiểm Tra (Cách 1)

```
int KiemTra_C1(int a[][MAXC], int m, int n)
{
   int i, j, dem = 0;

   for (i=0; i<m; i++)
      for (j=0; j<n; j++)
         if (LaSNT(a[i][j]==1) // là SNT
            dem++;

   if (dem == m*n)
      return 1;
   return 0;
}
```

Hàm Kiểm Tra (Cách 2)

```
int KiemTra_C2(int a[][MAXC], int m, int n)
{
   int i, j, dem = 0;

   for (i=0; i<m; i++)
      for (j=0; j<n; j++)
         if (LaSNT(a[i][j]==0)  // Không là SNT
            dem++;

   if (dem == 0)
      return 1;
   return 0;
}
```

Hàm Kiểm Tra (Cách 3)

```
int KiemTra_C3(int a[][MAXC], int m, int n)
{
   int i, j, dem = 0;

   for (i=0; i<m; i++)
      for (j=0; j<n; j++)
         if (LaSNT(a[i][j]==0) // Không là SNT
            return 0;

   return 1;
}
```

Tính tổng các phần tử

❖ Yêu cầu

- Cho trước mảng a, kích thước mxn. Tính tổng các phần tử trên:
  - Dòng d
  - Cột c

❖ Ý tưởng

- Duyệt mảng và cộng dồn các phần tử có tọa độ (dòng, cột) thỏa yêu cầu.

Hàm tính tổng trên dòng

```
int TongDong(int a[][MAXC], int m, int n, int d)
{
   int j, tong;

   tong = 0;

   for (j=0; j<n; j++)  // Duyệt các cột
      tong = tong + a[d][j];

   return tong;
}
```

Hàm tính tổng trên cột

```
int TongCot(int a[][MAXC], int m, int c)
{
   int i, tong;

   tong = 0;

   for (i=0; i<m; i++)  // Duyệt các dòng
      tong = tong + a[i][c];

   return tong;
}
```

Tìm giá trị lớn nhất của Mảng

❖ Yêu cầu

- Cho trước mảng a, kích thước mxn. Tìm giá trị lớn nhất trong mảng a (gọi là max)

❖ Ý tưởng : kỹ thuật đặt lính canh

- Giả sử giá trị max hiện tại là giá trị phần tử đầu tiên a[0][0]
- Lần lượt kiểm tra các phần tử còn lại để cập nhật max.

Hàm tìm giá trị lớn nhất của Mảng

```
int TimMax(int a[][MAXC], int m, int n)
{
   int i, j, max;

   max = a[0][0];  // lính canh

   for (i=0; i<m; i++)
      for (j=0; j<n; j++)
         if (a[i][j] > max)
            max = a[i][j];

   return max;
}
```

Lưu ý

❖ Liên hệ giữa chỉ số mảng 1 chiều và chỉ số mảng 2 chiều

- Giả sử mảng một chiều được nối tiếp theo từng dòng của mảng 2 chiều
- Mảng 2 chiều a có M dòng và N cột. (3 dòng và 4 cột), vị trí dòng d, cột c
- Mảng 1 chiều b M*N phần tử (có 3*4=12 phần tử), vị trí i
- Công thức chuyển đổi: từ 2 chiều sang 1 chiều (biết d và c, cần tính i)

a_{MxN}

1	21	30	4
5	3	10	20
-1	24	200	70

i = d*N+ c

- Công thức chuyển đổi: từ 1 chiều sang 2 chiều

(biết i, cần tính c và d)

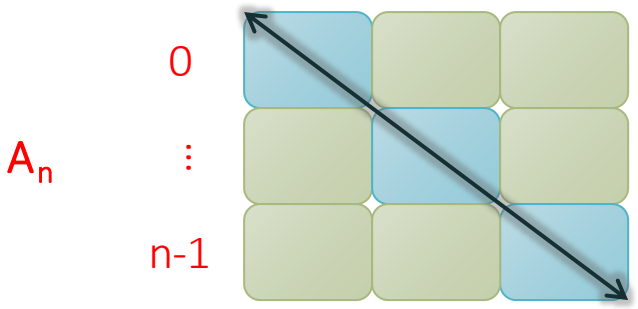
d = i / N
c = i % N

1	21	30	4	5	3	10	20	-1	24	200	70

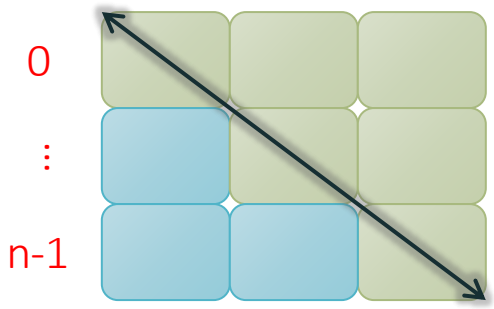
Lưu ý

- Mảng hai chiều có số dòng bằng với số cột: Gọi là ma trận vuông

Đường chéo chính

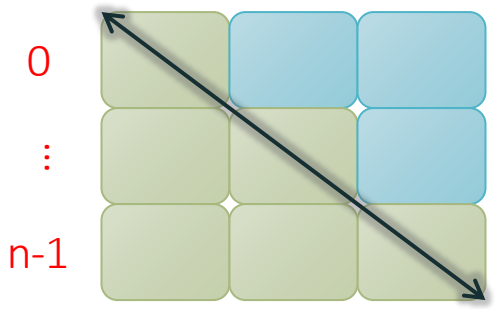


Ma trận tam giác dưới



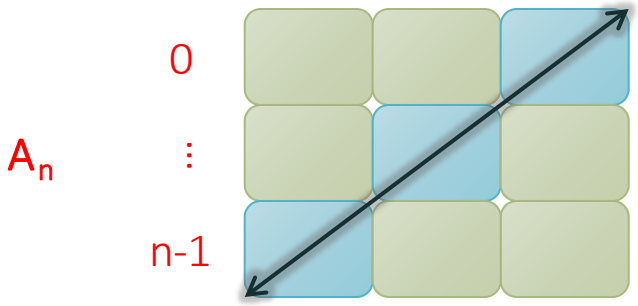
dòng > cột

Ma trận tam giác trên



dòng < cột

Đường chéo phụ



dòng + cột = n-1

Khai báo: int A[100][100];
float C[20][20];

Ví dụ

Nhập vào một ma trận vuông các số nguyên.

a) Tính tổng các phần tử trên đường chéo chính/phụ.

b) Tìm giá trị nhỏ nhất trong ma trận tam giác dưới.

c) Đếm số lượng giá trị chẵn trong ma trận tam giác trên

Ví dụ

- Tính tổng các phần tử trên đường chéo chính/phụ

```
int sumduongcheochinh(int  a[][MAX],int n)
{
        int tong =0;
        for (int i=0;i<n;i++)
                tong = tong + a[i][i];
        return tong;
}
```

```
int sumduongcheophu(int  a[][MAX],int n)
{
        int tong =0;
        for (int i=0;i<n;i++)
                tong = tong + a[i][n-1-i];
        return tong;
}
```

Ví dụ

- Tìm giá trị nhỏ nhất trong ma trận tam giác dưới (không tính các phần tử trên đường chéo chính)

Cách 1

```
int mintamgiacduoi(int a[][MAX], int n)
{
  int minvalue = INT_MAX;
  for (int i=0;i<n;i++)
     for (int j=0;j<n;j++)
       {
         if (j<i && a[i][j]<minvalue)
          minvalue = a[i][j];
       }
  return minvalue;
}
```

Cách 2

```
int mintamgiacduoi_2(int a[][MAX], int n)
{
   int minvalue = INT_MAX;
   for (int i=0;i<n;i++)
       for (int j=0;j<i;j++)
           {
          if (a[i][j]<minvalue)
               minvalue = a[i][j];
           }
    return minvalue;
}
```

Ví dụ

- Đếm số lượng giá trị chẵn trong ma trận tam giác trên. (không tính các phần tử trên đường chéo chính)

Cách 1

```
int counteventamgiactren(int a[][MAX], int n)
{
  int count = 0;
  for (int i=0;i<n;i++)
     for (int j=0;j<n;j++)
     {
        if ( j > i && a[i][j]%2==0)
       count++;
     }
   return count;
}
```

Cách 2

```
int counteventamgiactren_2(int a[][MAX], int n)
{
   int count = 0;
    for (int i=0;i<n;i++)
          for (int j=n-1;j>i;j--)
          {
        if (a[i][j]%2==0)
              count++;
          }
     return count;
}
```

Mảng nhiều chiều

- Khai báo mảng 3 chiều: int A[50][10][20];
  - Truy xuất các phần tử thông qua chỉ số: A[0][0][0], A[0][0][1], A[0][1][0], A[10][5][14]…

- Tổng quát mảng nhiều chiều:

<Kiểu_Cơ_Sở> <Tên_Biến_Mảng>[<N1>] [<N2>]…[<Nk>];

- Tuy nhiên, việc thao tác với những mảng nhiều chiều như thế này khá phức tạp

Các ví dụ minh họa

1. Nhập vào một ma trận các số nguyên.

a) Liệt kê chỉ số các dòng có chứa

- giá trị chẵn/lẻ trong ma trận các số nguyên
- số nguyên tố trong ma trận các số nguyên.

b) Kiểm tra một cột trong ma trận có giảm dần hay không?

c) Liệt kê các cột giảm dần.

d) Liệt kê cột có chứa nhiều chẵn nhất.

e) Liệt kê dòng có chứa nhiều số nguyên tố nhất

Các ví dụ minh họa

a) Liệt kê chỉ số các dòng có chứa giá trị lẻ trong ma trận các số nguyên

```
void findrow_odd_number(int a[][MAX],int m, int n)
{       int count = 0;
        for (int i=0; i< m; i++)
        {
                for (int j=0; j<n; j++)
                {
                        if (a[i][j]%2!=0) //
                        {
                                count++;
                                cout <<"dong:" <<i <<" co chua so le" <<endl;
                                break;
                        }
                }
        }
        if (count==0)
                cout<<"khong co dong nao chua so le" <<endl;
}
```

Các ví dụ minh họa

b) Kiểm tra một cột trong ma trận có giảm dần hay không?

```
int checkcolumndescending(int  a[][MAX],int m, int n, int k)
{
	int flag = 1; // 0: khong giam ; 1: giam dan
	for (int i=0;i<m-1;i++)
	{
		if(a[i][k]< a[i+1][k]) // khong giam dan
		{
			flag = 0;
			break;
		}
	}
	return flag;
}
```

Các ví dụ minh họa

c) Liệt kê các cột giảm dần

```
void lietkecolumndescending(int  a[][MAX],int m, int n)
{
	int check = 0;
	for (int k=0;k<n; k++) // duyet column
	{	//cap nhat lai flag sau moi buoc duyet theo cot
		int flag = 1;	// 0: khong giam ; 1: giam dan
		for (int i=0;i<m-1;i++)
		{
			if(a[i][k]<a[i+1][k]) // khong giam
			{	flag = 0;
				break;
			}
		}
		if (flag == 1)
		{	cout<<"column "<<k <<": co thu tu giam dan" <<endl;
			check = 1;
		}

	}
	if (check == 0)
		cout<<"khong co cot co thu tu giam dan" <<endl;
}
```

Các ví dụ minh họa

e) Liệt kê dòng có chứa nhiều số nguyên tố nhất

```
void lietkerowMAXprime(int a[][MAX],int m, int n)
{       int countprimes;      int indexrow = -1;    int maxprime = 0;
        for (int i=0;i<m;i++)
        {
                countprimes = 0;
                for (int j=0;j<n;j++)
                {
                        if(checkprime(a[i][j])==1)
                                countprimes++;
                }
                if (countprimes>maxprime)
                {       maxprime = countprimes;
                        indexrow = i;
                }
        }
        if (indexrow != -1)
        {       cout<<"\ndong chua nhieu SNT la:"<< indexrow <<endl;
                cout<<"\ntong so luong SNT:"<<maxprime<<endl;
        }
        else
                cout<<"\nkhong dong nao chua SNT" <<endl;
}
```

Bài tập mảng 1 chiều

1) Nhập vào mảng một chiều các số nguyên có N phần tử (N: nhập từ bàn phím)

a) Đếm số lần xuất hiện của các số nguyên dương.

b) Tính tổng tất cả các phần tử không âm.

c) Đếm số phần tử là số nguyên tố và tính tổng các phần tử này.

d) Đếm số phần tử là số chính phương và tính tổng các phần tử này.

e) Tìm số lớn nhất/ nhỏ nhất trong mảng

f) Tìm số nguyên tố lớn nhất/ nhỏ nhất (nếu có) trong mảng

Bài tập mảng 1 chiều

2) Nhập vào một mảng có N phần tử chứa các số nguyên

a) Kiểm tra xem mảng có phần tử trùng nhau không

b) Điếm số lần xuất hiện của các phần tử trong mảng

c) Xóa tất cả các phần tử trùng nhau (nếu có) trong mảng

d) Sắp xếp mảng tăng dần

e) Sắp xếp mảng giảm dần

f) Giả sử mảng đang được sắp tăng dần. Nhập số nguyên X, chèn X vào dãy đang có sao cho dãy vẫn được sắp xếp tăng dần, xuất dãy kết quả ra màn hình.

Bài tập mảng 1 chiều

3) Nhập vào 2 mảng một chiều các số nguyên

a) Nối 2 mảng một chiều thành một mảng

b) Xóa trên 2 mảng này tất cả các phần tử trùng nhau của 2 mảng

4) Nhập vào 2 mảng một chiều các số nguyên có cùng kích thước

a) Trộn 2 mảng một chiều thành một mảng một chiều với mỗi phần tử của mảng mới là tổng của 2 phần tử tương ứng từ 2 mảng cho trước.

b) Nhập vào 2 mảng có cùng kích thước, tạo mảng mới gồm các phần tử là UCLN của 2 phần tử tương ứng.

Bài tập mảng 1 chiều

5) **THCSLT6-01**

Cho dãy a gồm n số tự nhiên $a_o,a_1,a_2,...,a_{n-1}$. Hãy thực hiện các công việc sau:

a. Đếm số lượng số của dãy a thỏa điều kiện: hai số liền kề trái/kề phải với nó là số nguyên tố.

b. Tìm dãy b, biết số b_i là viết đảo ngược của số ai; ví dụ a_i =5334 thì b_i tương ứng là 4335.

c. Tìm dãy c, biết c_i là số lượng các số nguyên tố nhỏ hơn a_i; ví dụ a_i =18 thì c_i tương ứng là 7.

d. Tìm giá trị lớn thứ k của mảng (giả sử mảng có giá trị lớn thứ k).

e. Hãy đưa các số chẵn (ngoại trừ số 0) về đầu mảng, các số lẻ về cuối mảng, còn các số 0 ở giữa.

Bài tập mảng 1 chiều

6) **THCSLT6-02**

Cho mảng một chiều *a* chứa *n* số nguyên dương (kiểu unsigned long). Hãy thực hiện các công việc sau:

a. Tìm tổng các chữ số của tất cả các số của mảng *a*.

b. Tìm số nguyên tố lớn nhất của mảng *a*; nếu mảng *a* không có số nguyên tố nào thì xuất giá trị 0.

c. Tính tổng các ước số thực sự của mỗi số của mảng *a*. Ví dụ 6 có các ước số thực sự là 1,2,3

d. Tìm số nguyên tố nhỏ nhất của mảng *a*; nếu mảng *a* không có số nguyên tố nào thì xuất giá trị 0.

e. Cho biết số lượng các ước số của từng số của mảng *a*. Ví dụ 6 có 4 ước số là 1,2,3,6.

Bài tập: mảng 1 chiều

❖ Tìm kiếm và sắp xếp (Nâng cao)

1) Kiểm tra xem mảng có tăng dần hay giảm dần không.
2) Đếm số mảng con (các phần tử liên tiếp) tăng dần hoặc giảm dần trong mảng.
3) Cho mảng n phần tử và k < n. In ra tổng lớn nhất của k phần tử liên tiếp xuất hiện trên mảng.
4) Đếm số lượng các phần tử khác nhau xuất hiện trong mảng
5) **THCSLT6-08.**

   Cho hai dãy số nguyên $\{a\}$ và $\{b\}$, trong đó dãy $\{a\}$ có n phần tử và dãy $\{b\}$ có m phần tử. Các số trong các dãy này đã được sắp tăng dần. Hãy trộn các phần tử của hai dãy này thành một dãy $\{c\}$ sao cho dãy $\{c\}$ cũng được sắp tăng dần.

Bài tập: mảng 1 chiều

❖ Tìm kiếm và sắp xếp (Nâng cao)

6) Sắp xếp các phần tử trên mảng sao cho các số dương tăng dần và ở đầu mảng, các số âm giảm dần và ở cuối mảng, các số 0 ở giữa.
7) Sắp xếp các phần tử trên mảng sao cho các số chẵn tăng dần, các số lẻ giảm dần.
8) Sắp xếp các phần tử trên mảng sao cho các số chẵn tăng dần và ở đầu mảng, các số lẻ giảm dần và ở cuối mảng.
9) Kiểm tra xem có tồn tại mảng con tăng dần hay giảm dần không. Nếu có, in mảng con tăng dần dài nhất xuất hiện trong mảng. Nếu có nhiều mảng cùng dài nhất thì chỉ cần in ra một.

Bài tập mảng 2 chiều

1. Viết chương trình thực hiện các yêu cầu sau:

a) Tính tổng các số lẻ trong mảng 2 chiều a

b) Đếm số lượng các số chính phương trong mảng 2 chiều a

c) Tìm số nguyên tố lớn nhất trong mảng 2 chiều

d) Kiểm tra xem mảng có dòng nào chứa các số tăng dần hay không?

e) Sắp xếp các phần từ tăng dần theo mỗi dòng

f) Sắp xếp các phần tử tăng dần theo mỗi dòng đồng thời tăng dần theo mỗi cột.

Bài tập mảng 2 chiều

2. Nhập vào mảng 2 chiều các số nguyên có kích thước mxn (m dòng, n cột)

a) Tìm các giá trị nhỏ nhất trên mỗi dòng

b) Tìm giá trị lớn nhất trong các giá trị nhỏ nhất của mỗi dòng

c) Tìm giá trị lớn nhất của mỗi cột

d) Tìm giá trị nhỏ nhất trong các giá trị lớn nhất của mỗi cột

Bài tập mảng 2 chiều

3. Nhập vào mảng 2 chiều các số nguyên có kích thước mxn (m dòng, n cột)

a) Tìm các dòng có chứa số nguyên tố

b) Tìm cột chứa nhiều số chẵn nhất

c) Liệt kê các dòng chứa số âm

Bài tập mảng 2 chiều

CÂU 4 (2 điểm)

Cho mảng hai chiều *a* gồm *m* dòng *n* cột, các phần tử là các số nguyên dương. Hãy tạo mảng hai chiều *b* từ mảng *a*, sao cho b_{ij} là số nguyên tố nhỏ nhất lớn hơn hoặc bằng a_{ij} tương ứng.

CÂU 5 (2.0 điểm)

Cho mảng hai chiều a gồm *m* dòng *n* cột, các phần tử là các số nguyên dương.

Tìm mảng *b*, biết $b_{ij} = a_{ij} * k$ với *k* là giá trị nhỏ nhất của dòng *i* nếu i là số lẻ và $b_{ij} = a_{ij} * k$ với k là giá trị lớn nhất của dòng *i* nếu i là số chẵn.

Tìm mảng c, biết $c_{ij} = l$, với *l* là tổng số lượng số nguyên tố trên dòng i và trên cột j (nếu a_{ij} là số nguyên tố thì chỉ được đếm một lần trên dòng hoặc trên cột).